

REDIS

Open source in-memory data structure store

Riciputi Jacopo

30 Maggio 2018



Indice

① Redis

Cos'è

② Primi passi

Get started

Strutture Dati

Nodejs client

③ Utilizzo

Caching

Message Broker

Database

④ Casi d'uso

Clash of Clans

Waze

⑤ Conclusioni



Cos'è

Redis è uno store di strutture dati completamente in-memory, prestante e open source.

È utilizzato principalmente come:

- Database;
- Cache;
- Message broker.

All'interno di:

- applicazioni Web;
- videogiochi;
- sistemi IoT.



Perchè è uno dei più utilizzati

Nel mondo NoSQL Redis risulta essere **secondo** solo a MongoDB e 7° in classifica considerando i database relazionali.¹

I suoi **punti di forza**:

- La composizione chiave-valore permette di gestire più strutture dati;
- Il mantenimento dei dati in memoria e il linguaggio in cui è scritto (C) lo rendono **estramente veloce** e con tempi di latenza minimi;
- Opzionalmente permette di aggiungere **persistenza**;
- Supporta le transazioni;
- Fornisce funzionalità per eseguire **Lua scripting**, grazie a un interprete built-in;
- Offre **high availability** e supporta la suddivisione dei dati all'interno di una rete **cluster** di macchine.

¹<https://db-engines.com/en/ranking>

Estramente veloce

Why use it? Redis is Blazing Fast™!



“Why is a Ferrari™ so fast?”
Answer: **Performance dictates Design**

redislabs



Primi passi



Primi Passi

Installare Redis richiede qualche semplice passo se scaricato dalla pagina ufficiale² oppure è possibile ottenere una versione completa su Debian o derivate lanciando il comando ***sudo apt-get install redis-server***.

- Una volta installato, con il comando ***redis-server*** si avvia un'istanza sulla porta di default (6379)
- È possibile collegarsi da terminale a questa istanza tramite il client ***redis-cli***

A questo punto si è completamente operativi.

Esempio base:

Redis per la gestione di semplici chiavi fornisce i comandi **SET** e **GET**

- **SET** richiede una chiave e il corrispondente valore.
Es. SET foo bar → OK
- **GET** Data una chiave ne ritorna il valore. Es. GET foo → bar

²<https://redis.io/download>

Strutture Hash

HSET consente di creare una chiave a cui associare più coppie chiave-valore.

> HSET hfoo firstKey firstValue

(integer 1)

> HSET hfoo secondKey secondValue

(integer 1)

È possibile riottenere questi valori in differenti modi:

- HGET hfoo firstKey → "firstValue"
- HGETALL hfoo
1) "firstKey" 2) "firstValue" 3) "secondKey" 4) "secondValue"
- **HKEYS** per ottenere tutte le chiavi
- **HVALS** per ottenere tutti i valori



Liste

Redis offre la possibilità di gestire le liste, tramite i comandi **LPUSH** e **LPOP** per le operazioni in testa, **RPUSh** e **RPOP** per quelle in coda.

```
> LPUSH list 2  
(integer) 1  
> RPUSh list 3  
(integer) 2  
> LPUSH list 1  
(integer) 3
```

Redis restituirà i valori in ordine "1", "2", "3".
Si può verificare con l'istruzione **LRANGE**

```
> LRANGE list 0 -1  
1) "1"  
2) "2"  
3) "3"
```



Set

Due comandi principali, **SADD** e **SMEMBERS** rispettivamente per aggiungere valori a una chiave e visualizzare i suoi elementi.

```
> SADD myset 1 2 3
( integer ) 3
> SMEMBERS myset
1. "3"
2. "1"
3. "2"
```

Decisamente più interessanti i **Sordet Set**.



Sorted Set

Questa struttura dati è gestita da Redis associando a ogni valore uno **score**, sul quale sarà eseguito l'ordinamento.

Se per due valori coincide sarà eseguito sul valore lessicografico di quest'ultimi.

In questo caso i comandi base sono:

- **ZADD**: Prende in ingresso chiave, score, valore e li immagazzina
- **ZRANGE**: Data una chiave, indice di partenza e di fine ne visualizza in ordine **crescente** i valori;
- **ZREVRANGE**: Data una chiave, indice di partenza e di fine ne visualizza in ordine **decrescente** i valori;
- **ZRANGEBYLEX**, **ZREVRANGEBYLEX**, **ZREMRANGEBYLEX** e **ZLEXCOUNT** per manipolarli in base al valore lessicografico.



Sorted Set

```
> ZADD classe 1996 "Mario Rossi"  
( integer ) 1  
> ZADD classe 1993 "Fabio Bianchi"  
( integer ) 1  
> ZADD classe 1987 "Roberto Verdi"  
( integer ) 1  
> ZADD classe 1990 "Carlo Neri"  
( integer ) 1  
  
> ZRANGE classe 0 -1  
1) "Roberto Verdi"  
2) "Carlo Neri"  
3) "Fabio Bianchi"  
4) "Mario Rossi"
```



Nodejs client

npm install ioredis - Client molto semplice per Nodejs

```
var Redis = require("ioredis");
var redis = new Redis({
  port: 6379,
  host: "127.0.0.1",
  family: 4
});

redis.set("foo", "bar");
redis.get("foo", function(err, result) {
  console.log(result);
});

redis.del("foo");
redis.get("foo", function(err, result) {
  console.log(result);
});
```



Nodejs client

Output:

```
bar  
null
```

Qualcosa di più "complicato":

```
redis.hmset("me",  
            "name", "jacopo",  
            "surname", "riciputi",  
            "age", 22);  
  
redis.hgetall("me", function(err, result){  
    console.log(result);  
});
```

Output:

```
{ name: 'jacopo', surname: 'riciputi', age: '22' }
```



Utilizzo

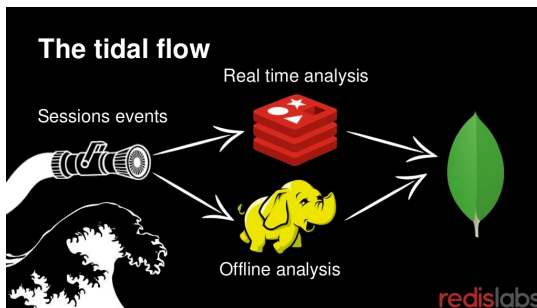


Caching

Problemi:

- Difficoltà nella digestione dei dati;
- Mobile, IoT, web 2.0 producono continuamente informazioni;
- Impossibile eseguire processi real-time.

Soluzione → Pulire i dati con Redis!!



Caching

L'idea è di accompagnare la sessione dell'utente con Redis.

Al suo termine i dati saranno processati, "ripuliti" e passati a contenitori più grandi (es. MongoDB)

Le sessioni terminano dopo un logout o un timeout.

- **Facile** identificare il logout
- **Difficile** rilevare un timeout, soprattutto considerando le migliaia di sessioni che possono essere attive

Qui Redis viene in soccorso!

Permette infatti di associare una scadenza alle chiavi e di gestirne le notifiche.



Message Broker

Redis implementa un **PUB/SUB messaging paradigm**.

Sarà possibile perciò avere client iscritti a un determinato canale e publisher che inviano messaggi.

Ogni messaggio avviene in broadcast!

- Iscrizione con **SUBSCRIBE foo**
- Pubblicazione con **PUBLISH foo hello**
- È possibile utilizzare pattern con **PSUBSCRIBE h?llo**
Accetterà messaggi da hello, hallo, hxllo ma non da hllo.



Database

Il mondo odierno necessita di **analisi istantanee**, i tradizionali tool richiedono ore, se non giorni!

Redis offre:

- Analisi in tempo reale;
- Transazioni atomiche;
- Supporto a un elevatissimo numero di transazioni per secondo;
- Scalabilità data l'assenza di struttura e la semplicità nella modellazione dei dati;
- Ulteriore scalabilità grazie alla gestione **cluster** integrata.



Casi d'uso



Clash of Clans

- Gioco real-time con milioni di utenti;
- Il datastorage principale → MongoDB;
- Utilizza Redis per il salvataggio e la progressione delle partite;
- Nel dettaglio per l'aggiornamento delle risorse real-time, i punteggi e le classifiche;



Waze

- Focus sul traffico real-time;
 - Più di **10 milioni** di utenti collegati nelle ore di punta;
 - Database principale → MongoDB;
 - Utilizza Redis per gli aggiornamenti in tempo reale su traffico, veicoli e i passeggeri;
- Appoggio temporaneo dei dati. Successivamente saranno elaborati per **aggiustamenti** e **notifiche**.



Conclusioni



In sintesi

- Prestazioni e facilità d'uso;
- Molteplici strutture dati;
- Scalabilità;
- Forte community, driver per molti linguaggi;
- Vari scenari: Caching, Message Broker, Database;
- Open source ed estendibile: RedisSearch / redisSQL / ReJSON / Redis Graph;
- Cuore italiano ! ³



³<https://github.com/antirez>